

The Promise of Polynomial-based Local Search to Boost Boolean MAX-CSP Solvers

Christine D. Hang Ahmed Abdelmeged Daniel Rinehart Karl J. Lieberherr

Northeastern University,
College of Computer and Information Science,
360 Avenue of the Arts, Boston, MA 02115, USA
{christine,mohsen,danielr,lieber}@ccs.neu.edu

Abstract. We propose a novel family of polynomial-based local search algorithms for MAX-CSP. From this family, we present an optimal, fast algorithm, called ELS (Evergreen local search). We evaluate ELS as a preprocessor for state-of-the-art MAX-SAT solvers on a small benchmark which shows promising speed-up or improvement in quality. To entertain the reader, we derive ELS from a simple game, called Evergreen, which is about creating and solving CSP-foemulae.

1 Introduction

Is backtracking search a “must-have” to boost the performance of MAX-CSP solvers? To answer “no” to this question, we introduce a novel polynomial-based local search algorithm and use it as a preprocessor to boost Boolean MAX-CSP solvers. To entertain the reader, we derive this algorithm from a MAX-CSP-based two-player game called Evergreen.

We define the following terminologies. A Boolean CSP formula is a non-empty bag of constraints, each of which consists of an integer multiplicity and a Boolean relation of some rank r . The multiplicity indicates how often the constraint appears in the bag, and the Boolean relation of rank r represents a Boolean formula involving r variables. From now on, we simply refer to Boolean CSP formula by CSP formula. An assignment for a CSP formula F maps the variables of F to Boolean values. $fsat(F, J)$ is the fraction of satisfied constraints in formula F under assignment J , where a satisfied constraint is one whose corresponding Boolean formula evaluates to true under J .

We further define a constraint language Γ as a set of relations, and a $CSP(\Gamma)$ formula as a CSP formula that only contains relations in Γ . Then, a $MAX-CSP(\Gamma)$ problem can be formulized as one that has as input a $CSP(\Gamma)$ formula F and as output an assignment J such that $fsat(F, J)$ is maximized. Boolean MAX-CSP solvers are used to solve $MAX-CSP(\Gamma)$ problems.

We propose to use polynomials as a heuristic in local search, and construct a polynomial-based booster for MAX-CSP solvers. To give the intuition, we create the Evergreen game (played by Alice and Bob) to demonstrate not only how polynomials can be derived from $MAX-CSP(\Gamma)$ problems, but also how they can be utilized to generate boosters for MAX-CSP solvers. We illustrate this presentation scheme in figure 1.

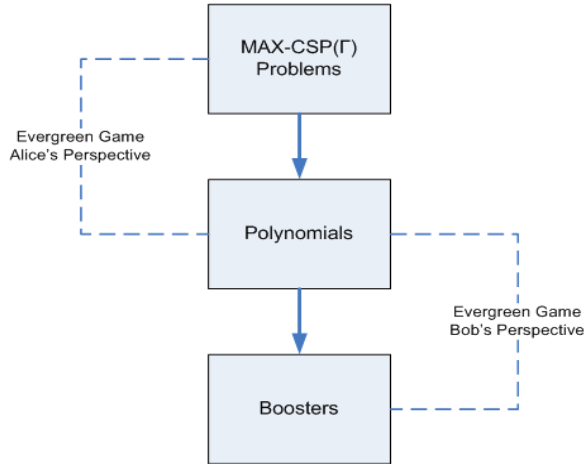


Fig. 1. Presentation Scheme

1.1 Contributions

The paper makes the following contribution:

Technology derivation with potential MAX-CSP solver impact.

The simple Evergreen Game acts as the generator of technology that might be useful for MAX-CSP solvers in general. The Evergreen Game relies on polynomials that abstractly represent CSP formulae. These polynomials lead to P-optimal algorithms [1]: An algorithm for $MAX-CSP(\Gamma)$ is P-optimal if it guarantees to satisfy a fraction τ_Γ of all constraints and the set of formulae in which the fraction $\tau_\Gamma + \epsilon$ ($\epsilon > 0$) can be satisfied, is NP-complete.

1.2 Paper Organization

The rest of the paper is organized as follows: Section 2 introduces the skeleton of our local search algorithm *ELS*. Section 3 presents the Evergreen Game played by Alice and Bob. Section 4 analyses the game from Alice’s perspective and derives the polynomials used in *ELS*. Section 5 analyses the game from Bob’s perspective and completes *ELS*. Section 6 postulates two “laws” to which future MAX-CSP solvers should conform, and introduces *ELS*’ role of enforcing these two “laws”. Section 7 illustrates the boosting effect of *ELS* on MAX-CSP solvers. Sections 8 and 9 conclude this paper.

2 Evergreen Local Search

In local search, a neighbourhood relation is used to define the set of assignments that are considered neighbours of a given assignment. We introduce the notion of flipping a variable as setting it to the negation

of its current value. An assignment H_2 is a k -flip of assignment H_1 , if k variables in H_2 is flipped with respect to H_1 . For a given formula F and a given assignment J_1 , we consider an assignment J_2 to be in the evergreen neighbourhood (EN) of J_1 , if J_2 has a satisfaction ratio that is no less than the average of the satisfaction ratios of all k -flips of J_1 . It is important to recognize that J_2 does not have to be a k -flip of J_1 .

For a given assignment J , traditional k -opt local search algorithms consider all the k -flips of J to be its neighbours. What distinguishes our local search algorithm from those traditional algorithms is that we only pick the assignments that are at least as good as the average of the k -flips of J to be its neighbours. Specifically, let $lap(F, J, k)$ be the mean fraction of satisfied constraints over the $\binom{n}{k}$ assignments for F where among the n variables of J exactly k of them are flipped. We will introduce how to calculate $lap(F, J, k)$ in the context of the Evergreen game in section 4.

Definition 1. For formula F , J_2 is a k -evergreen-neighbour of J_1 , written as $EN(F, J_1, J_2, k)$, if $fsat(F, J_2) \geq lap(F, J_1, k)$.

We define k -evergreen-neighbour finder, $ENF(F, J_1, k)$, as an algorithm that takes as inputs a formula F , an assignment J_1 and an integer $k(0 \leq k \leq n)$, and returns as output an assignment J_2 such that $EN(F, J_1, J_2, k)$ holds. For a given formula, a given assignment and a given integer k , there exists at least one k -evergreen-neighbour. Note that $ENF(F, J_1, k)$ specifies one local search step.

Let k_{max} be the very $k(0 \leq k \leq n)$ that maximizes $lap(F, J, k)$ for a given formula F and a given assignment J . If we fix the k of $ENF(F, J_1, k)$ to be k_{max} , we will derive a maximal evergreen-neighbour finder, $mENF(F, J_1)$, which takes as inputs a formula F and an assignment J_1 , and returns as output an assignment J_2 such that $EN(F, J_1, J_2, k_{max})$ holds. We will specify this maximal local search step in the context of the Evergreen game in section 5.

We construct our evergreen local search algorithm, ELS , employing a maximal evergreen-neighbour finder at each step until the fraction of satisfied constraints stops increasing.

```

ELS(F, J1)
1  new ← fsat(F, J1)
2  repeat
3      J2 ← mENF(F, J1)
4      old ← new
5      new ← fsat(F, J2)
6      J1 ← J2
7  until old = new

```

3 The Evergreen Game

The Evergreen Game is a two-player game focused on constructing and solving Boolean MAX-CSP problems. The players win and lose money

based on how good they solve those problems. Let $Evergreen(r, s)$ define a class of games where r is the maximum rank of relations used in the formulae and s is the number of distinct relations used in a formula.

The game consists of a setup phase and two rounds. In the setup phase the players jointly decide: the set of s distinct relations to use (observing the r maximum rank restriction), the total number of variables that can be used (denoted as n), which player will go first, the wager, and an action time-limit. Each round consists of a formula construction action, an assignment production action, and a money settlement phase. During the formula construction action the first player constructs a CSP formula that uses no more than n variables, that contains at least one constraint, and that consists only of relations from the agreed upon set. Duplication of a variable in a relation is not permitted. At or before the action time-limit the first player gives the constructed formula to the second player. During the assignment production action the second player finds an assignment that satisfies as many constraints as possible in the received formula. At or before the action time-limit the second player gives the assignment to the first player. In the money settlement phase the first player pays the second player the percentage of the wager equal to the percentage of constraints satisfied by the second player's assignment when applied to the formula created by the first player. Should either player fail to construct a valid formula or assignment the entire wager is forfeit to the other player and the game ends immediately. The second round is played the same as the first, but with the player order reversed. Throughout this paper, we choose Alice and Bob to be the two players, and let Alice goes first. As an example, Alice and Bob are playing the game with a wager w of 1 million dollars and the constraints limited to $\Gamma = \{OR(x, y), NOT(x)\}$. Alice starts by constructing $F_{Initial} = \{100 : NOT(x), 150 : NOT(y), 200 : OR(x, y)\}$. Bob tries to find an assignment that satisfies the largest possible fraction of constraints. For example, the assignment $\{x = true, y = false\}$ will satisfy $(150 + 200)/450 \approx 0.78$. Alice then pays Bob 0.78 million dollars ($w * 0.78$). Bob now constructs a formula that Alice solves and pays Alice the percentage of the wager that she solved.

In this case, Alice's first move ($F_{Initial}$) was not optimal. She could have chosen another formula in which a smaller percentage could have been satisfied by Bob, saving her money. Some important questions about the Evergreen Game are: How can Alice limit the amount she has to pay to Bob? How should Alice construct the formula? How can Bob quickly satisfy many constraints? For the purpose of this paper, we are most interested in answering the third question.

4 Alice's Perspective

Alice's objective is to minimize her loss. Therefore, she first needs to assume that for any CSP formula she gives Bob, he can always find the best assignment that achieves the maximum satisfaction ratio. Then, she should look for a CSP formula F whose maximum satisfaction ratio is

the closest to the minimum. We formally state Alice's objective as:

$$\tau_\Gamma = \inf_{\text{CSP formula: } F} \max_{\text{Assignment: } J} \text{fsat}(F, J)$$

At a first glance, this problem involves searching through all CSP formulae that Alice is allowed to construct. However, we show that Alice can cut her search space drastically by searching only for symmetric formulae.

4.1 Symmetric Formulae

Let π_n be the full permutation group on the n variables of F . For every $\sigma \in \pi_n$ let $\sigma(F)$ be the permuted formula, which is the result of substituting $\sigma(x)$ for all variables x in F .

Definition 2. *A CSP formula is called symmetric if any permutation of the variables in the formula returns the same formula up to a permutation of the constraints.*

Corollary 1. *If F is a symmetric CSP formula then for all permutations σ in π_n and all assignments J of F : $\text{fsat}(F, J) = \text{fsat}(\sigma(F), J) = \text{fsat}(F, \sigma^{-1}(J))$*

Lemma 1. *Let F be an asymmetric CSP formula. Symmetrize F by using the full permutation group on the n variables of F . Call these permutations $\sigma_1 \dots \sigma_{n!}$. $\text{sym}(F)$ is the concatenation of $\sigma_1(F) \dots \sigma_{n!}(F)$ and has $n! \cdot \text{constraints}(F)$ constraints. For every assignment J to F the following holds:*

$$\text{fsat}(\text{sym}(F), J) = \frac{1}{n!} \cdot \sum_{i=1}^{n!} \text{fsat}(\sigma_i(F), J)$$

Theorem 1. *For every CSP formula F , the symmetrized formula $\text{sym}(F)$ satisfies:*

$$\max_{\text{Assignment: } J} \text{fsat}(\text{sym}(F), J) \leq \max_{\text{Assignment: } J} \text{fsat}(F, J)$$

Proof. The proof is best explained in terms of a two-dimensional matrix (shown in table 1) for a given CSP formula F with n variables like the one shown below. The rows of the matrix correspond to the 2^n assignments for F and the columns correspond to formulae, namely to the $n!$ permutations all applied to F . The first permutation is the identity and we add one more column (the last one) to the matrix for the symmetrized formula $\text{sym}(F)$. An entry in the matrix gives the fraction of satisfied constraints by the assignment (row) to the permuted formula (column). According to lemma 1 and the fact that if the mean of a set of numbers is f then at least one number is greater than or equal to f , the last entry of every row is less than or equal to one of the entries x in the same row. Let the column where this entry resides correspond to permutation σ_j . We construct the inverse of σ_j (because $\text{fsat}(\sigma(F), a_i) = \text{fsat}(F, \sigma^{-1}(a_i))$) and apply it to a_i . This gives us a new row corresponding to $\sigma_j^{-1}(a_i)$

Table 1. Matrix

	$\sigma_1(F)$	\dots	$\sigma_j(F)$	\dots	$\sigma_{n!}(F)$	$sym(F)$
a_1						
\vdots						
a_i			x			$\leq x$
\vdots						
$\sigma_j^{-1}(a_i)$	x					
\vdots						
a_{2^n}						

and the claim is that in that row there is also the same entry x , namely in column 1 (identity permutation).

The above argument shows that

$$\forall F \forall J \exists J' : fsat(sym(F), J) \leq fsat(F, J'),$$

where F is a CSP formula and J and J' are assignments for $sym(F)$ and F respectively. Now we choose the maximum assignment J_{maxsym} for $sym(F)$. Also for this assignment the inequality holds. Therefore, F must have an assignment that is at least as good as J_{maxsym} for $sym(F)$. Hence, the theorem follows.

Now, Alice's objective is simplified and can be restated as:

$$\tau_\Gamma = \inf_{Symmetric\ CSP\ formulae:F} \max_{Assignment:J} fsat(F, J)$$

This is correct because theorem 1 states that for every asymmetric formula there exists a symmetric one which is worse. It is sufficient to minimize among the symmetric formulae because in the asymmetric ones larger fractions can be satisfied which do not influence the minimum.

Recall Anna's first move, $F_{Initial}$, in section 3. According to definition 2, $F_{Initial}$ is not symmetric, which is why we stated that Alice's first move was not optimal.

For the purpose of this paper, we only derive the polynomials needed to generate our local search algorithm. We elaborate Alice's best strategy in our technical report [2].

4.2 Deriving Polynomials

We first derive mean polynomials, analyse them in terms of symmetric formulae, and then construct from mean polynomials the look-ahead polynomials (*lap*) that are used in our local search algorithm.

Mean Polynomials We define $mean_F(n, k)$ to be the average fraction of satisfied constraints in F among all assignments to the n variables

of F that set exactly k variables to true. By elementary combinatorial analysis, we get the following:

$$\begin{aligned} \text{mean}_F(n, k) &= \sum_{i=1}^s t_{R_i}(F) \cdot \text{SAT}_{R_i}(n, k) \\ \text{SAT}_{R_i}(n, k) &= \frac{\sum_{j=0}^r \frac{q_j(R_i)}{\binom{r}{j}} \cdot \binom{k}{j} \cdot \binom{n-k}{r-j}}{\binom{n}{r}} \end{aligned}$$

where $t_{R_i}(F)$ is the fraction of constraints in F that contain relation R_i , s is the number of relations in Γ , r is $\text{rank}(R_i)$, and $q_j(R_i)$ is the number of satisfied rows in the truth table of relation R_i when j variables are set to true.

Theorem 2. *If F is a symmetric CSP formula then:*

$$\max_{\text{all assignments } J \text{ for } F} \text{fsat}(F, J) = \max_{0 \leq k \leq n} \text{mean}_F(n, k)$$

Proof. According to corollary 1, permuting an assignment doesn't change the fraction of satisfied constraints in a symmetric formula. In other words, for a symmetric formula all that matters in an assignment is the number of true variables. Since the mean polynomial averages over assignments that set only k variables to true, the fraction of satisfied constraints predicted by $\text{mean}_F(n, k)$ is exact.

According to theorem 2, Alice's search space can be reduced further using *mean* polynomials. This reduces the search space exponentially from size 2^n to n and it can be reduced even further by using calculus.

Look-ahead Polynomials We define $n\text{-map}(F, M)$ as a function that takes a CSP formula F and an assignment M and replaces each variable in F with its complement only if the variable is assigned to *true* in M . The name *n-map* comes from [3]. We assume formula F has n variables in total. We further assume that Γ is closed under n -mapping. If this is not the case then we can use its closure under n -mapping.

$$\text{lap}(F, J, k) = \text{mean}_{n\text{-map}(F, J)}(n, k)$$

We call this polynomial the look-ahead polynomial because it looks ahead into the search space.

5 Bob's Perspective

Bob has an ideal gambling strategy by which he never loses money. The key of such a strategy lies in both constructing an efficient algorithm A which guarantees to satisfy τ_Γ of any formula and giving back to Alice the hardest formula to solve.

Let the formula Bob receives from Alice be F_{Alice} , and let the fraction of F_{Alice} satisfied in polynomial time by applying algorithm A be $f_{\text{Alice}} =$

$A(F_{Alice})$, and let the fraction of the formula Bob gives back to Alice satisfiable in polynomial time be f_{Bob} . Therefore, we can represent Bob's gain as $(f_{Alice} - f_{Bob}) \cdot w$, where w is the wager. f_{Bob} is the minimum over all formulae, therefore $f_{Bob} \leq f_{Alice}$. If Bob can construct a polynomial time algorithm which guarantees to satisfy the fraction f_{Alice} of the formula that he receives from Alice, he will never lose money.

We introduce two gambling tools for Bob, a randomized algorithm by which with high probability Bob will not lose money and a derandomized algorithm which gurantees Bob will not lose money. We will use the latter to generate each Evergreen local search step of algorithm *ELS*.

5.1 Randomized Algorithm

Let F be the formula Bob receives from Alice. The randomized algorithm *RANDOMIZED-GAMBLER*(b) goes through all the variables and sets each of them to *true* with probability b . Intuitively, the more iterations we run this algorithm, the higher the probability will be. We first introduce this randomized algorithm and then bound the number of iterations we need in order to achieve a given probability.

```

RANDOMIZED-GAMBLER( $b$ )
1  bend a coin according to  $b$ 
2   $J \leftarrow \emptyset$ 
3  for each variable  $x \in F$ 
4      do flip the bent coin
5      if it is head
6          then  $J \leftarrow J \cup x$ 
7          else  $J \leftarrow J \cup \neg x$ 
8  return  $J$ 

```

For an arbitrary formula F containing c constraints, we denote by f_{avg} the satisfiable fraction that the *appmean* polynomial predicts, i.e. $f_{avg} = \text{appmean}_F(x)$ for some $0 \leq x \leq 1$. We denote by p the probability that after a single iteration the *RANDOMIZED-GAMBLER* finds an assignment satisfying a fraction greater than f_{avg} .

We first compute a lower bound of p . Consider the worst senario in which among the 2^n possible assignments of F , all whose satisfaction ratio is above f_{avg} satisfies exactly c constraints, whereas all whose satisfaction ratio is below f_{avg} satisfies exactly $c \cdot f_{avg} - 1$ constraints. We denote the probability in this worst senario by p_w . Thus, $p \geq p_w$.

$$c \cdot f_{avg} = p_w \cdot c + (1 - p_w) \cdot (c \cdot f_{avg} - 1)$$

$$p_w = \frac{1}{1 + c \cdot (1 - f_{avg})}$$

The probability that the *RANDOMIZED-GAMBLER* finds after n iteration an assignment satisfying a fraction greater than f_{avg} is $\delta = 1 - (1 - p)^n$. Since $1 - (1 - p)^n \geq 1 - (1 - p_w)^n \geq 1 - e^{-np_w}$, therefore, one needs to

make the number of iterations satisfy the following condition in order to achieve a given probability δ .

$$\begin{aligned}
1 - e^{-np_w} &\geq \delta \\
n &\geq -\frac{1}{p_w} \cdot \ln(1 - \delta) \\
n &\geq -(1 + c \cdot (1 - f_{avg})) \cdot \ln(1 - \delta)
\end{aligned}$$

A special case is when we set b to k_{max}/n , where k_{max} is the very k that maximizes the polynomial $mean_F(n, k)$, then the RANDOMIZED-GAMBLER will find with high probability an assignment that satisfies a fraction no less than the maximum of what the mean polynomial predicts. We name the randomized algorithm in this situation as EVERGREEN-GAMBLER and introduce it as follows.

```

EVERGREEN-GAMBLER( $F$ )
1   $b \leftarrow k_{max}/n$ 
2  RANDOMIZED-GAMBLER( $b$ )

```

5.2 Derandomized Algorithm

Polynomial-Based Derandomization We define REDUCE(l, F) as a function that takes a literal, l , and a formula, F , and produces a new formula which is the same as F with the variable corresponding to l assigned *true* if l is positive and assigned *false* otherwise. The derandomized algorithm is a deterministic algorithm which guarantees to satisfy the maximum fraction the mean polynomial predicts in polynomial time. We reconcile this algorithm from [1, 4]. Let F be the formula Bob receives from Alice.

```

EVERGREEN-PLAYER( $F$ )
1   $k \leftarrow 0, tm \leftarrow mean_F(n, t)$ 
2  for  $t \leftarrow 1$  to  $n$ 
3      do if  $mean_F(n, t) > tm$ 
4          then  $k \leftarrow t, tm \leftarrow mean_F(n, t)$ 
5   $J \leftarrow \emptyset$ 
6  for each variable  $x \in F$ 
7      do
8           $F_1 \leftarrow REDUCE(x, F)$ 
9           $F_0 \leftarrow REDUCE(\neg x, F)$ 
10     if  $mean_{F_1}(n - 1, k - 1) > mean_{F_0}(n - 1, k)$ 
11         then  $J \leftarrow J \cup x, k \leftarrow k - 1, F \leftarrow F_1$ 
12         else  $J \leftarrow J \cup \neg x, F \leftarrow F_0$ 
13  return  $J$ 

```

Intuitively, $mean_F(n, k)$ can be decomposed into $mean_{F_1}(n-1, k-1)$ and $mean_{F_0}(n-1, k)$. Among the $\binom{n}{k}$ cases in which exactly k variables are set to *true*, $\binom{n-1}{k-1}$ cases belong to the former, whereas $\binom{n-1}{k}$ cases belong to the latter. Therefore, we can derive the following recurrence relation for all $0 < k \leq n$.

$$mean_F(n, k) = \frac{\binom{n-1}{k-1}}{\binom{n}{k}} \cdot mean_{F_1}(n-1, k-1) + \frac{\binom{n-1}{k}}{\binom{n}{k}} \cdot mean_{F_0}(n-1, k)$$

for $k = 0$,

$$mean_F(n, 0) = mean_{F_0}(n, 0)$$

for $k = -1$,

$$mean_F(n, -1) = 0$$

Observe that $\frac{\binom{n-1}{k-1}}{\binom{n}{k}}$ and $\frac{\binom{n-1}{k}}{\binom{n}{k}}$ sum up to 1 by Pascal's rule, so for all $0 < k \leq n$, $mean_F(n, k) \leq \max\{mean_{F_1}(n-1, k-1), mean_{F_0}(n-1, k)\}$. Therefore, the assignment J , which algorithm EVERGREEN-PLAYER produces, has the following property

$$fsat(F, J) \geq \max_{0 \leq t \leq n} \{mean_F(n, t)\}$$

5.3 Generating Evergreen Local Search Steps

It is important to recognize that if we n -map a CSP formula F with respect to the *all false* assignment, we get back F itself, namely $F = n\text{-map}(F, \text{all false})$. According to the definition of look-ahead polynomials in section 4, we have

$$lap(F, \text{all false}, k) = mean_{n\text{-map}(F, \text{all false})}(n, k) = mean_F(n, k)$$

If we fix k to be k_{max} , which maximizes the polynomial $mean_F(n, k)$, we get

$$lap(F, \text{all false}, k_{max}) = mean_F(n, k_{max}) = \max_{0 \leq t \leq n} \{mean_F(n, t)\}$$

Therefore, the satisfaction ratio that EVERGREEN-PLAYER achieves has the following property

$$fsat(F, J) \geq lap(F, \text{all false}, k_{max})$$

This means that what EVERGREEN-PLAYER produces is indeed a maximal evergreen-neighbour of the *all false* assignment. We formalize this notion as:

$$\text{EVERGREEN-PLAYER}(F) = mENF(F, \text{all false})$$

Reversely, we can also generate each maximal local search step ($mENF$) by composing n -mapping and EVERGREEN-PLAYER, thus completing our local search algorithm *ELS*.

```

mENF( $F, J_1$ )
1   $F' \leftarrow n\text{-map}(F, J_1)$ 
2   $J_{aux} \leftarrow \text{EVERGREEN-PLAYER}(F')$ 
3   $J_2 \leftarrow J_1 \text{ xor } J_{aux}$ 
4  return  $J_2$ 

```

6 Implications of Evergreen on MAX-CSP Solvers

The insights from the Evergreen Game offer opportunities to improve MAX-CSP solvers, both complete solvers that provide a proof, and incomplete solvers, like stochastic local search solvers. The Evergreen Game shows that a non-trivial level of satisfaction can be reached in polynomial time. We would like to postulate two properties that future MAX-CSP solvers should have and that the designers of the solvers should be able to prove. If a MAX-CSP solver satisfies these laws it should have better performance on practically useful formulae.

6.1 Evergreen Law: Loss-Free

We postulate that future MAX-CSP solvers should guarantee, on their first try to construct an assignment, at least at the level of satisfaction that bounds Alice’s loss. This level of satisfaction must be reached quickly, i.e., in quadratic time in the size of the CSP formula. This can be achieved either by the probabilistic algorithm Evergreen-Gambler or its derandomized version Evergreen-Player.

6.2 Evergreen Law: Maximal

As an iterative application of the Loss-Free law, we postulate that future MAX-CSP solvers should guarantee to find a maximal assignment (defined below) after constructing at most c assignments, where c is the total number of constraints.

We consider an assignment M as maximal for a given CSP formula F , if

$$\max_{0 \leq k \leq n} \text{mean}_{n\text{-map}(F, M)}(n, k) = \text{mean}_{n\text{-map}(F, M)}(n, 0)$$

Note that if an assignment is not maximal, it cannot be a maximum assignment. A maximal assignment is not globally maximum. It is locally maximum in the sense that changing it with a maximum bias probability will not give a better assignment. Depending on Γ , finding a maximum assignment for a $\text{CSP}(\Gamma)$ formula can be NP -hard. On the other hand, finding a maximal assignment is always in P . The following algorithm finds a maximal assignment for a given CSP formula.

```

AGGRESSIVE-EVERGREEN-PLAYER( $F$ )
1   $A \leftarrow all\ false$ 
2   $newratio \leftarrow fsat(F, A)$ 
3  repeat
4       $M \leftarrow EVERGREEN-PLAYER(F)$ 
5       $oldratio \leftarrow newratio$ 
6       $newratio \leftarrow fsat(F, M)$ 
7       $F \leftarrow n\text{-map}(F, M)$ 
8       $A \leftarrow A\ xor\ M$ 
9  until  $oldratio = newratio$ 
10 return  $A$ 

```

Claim. The loop invariant of AGGRESSIVE-EVERGREEN-PLAYER(F) is $oldratio \leq newratio$.

In order to prove this loop invariant, we start by proving the following property of our Evergreen player.

Property 1. EVERGREEN-PLAYER returns an assignment which is at least as good as the *all false* assignment, namely

$$fsat(F, EVERGREEN-PLAYER(F)) \geq fsat(F, all\ false)$$

Proof. By the definition of the EVERGREEN-PLAYER algorithm, we have

$$\max_{0 \leq t \leq n} mean_F(n, t) \leq fsat(F, EVERGREEN-PLAYER(F))$$

Also, by the definition of *maxmean*, we have

$$mean_F(n, 0) \leq \max_{0 \leq t \leq n} mean_F(n, t)$$

Since $fsat(F, all\ false) = mean_F(n, 0)$, therefore, we have

$$fsat(F, EVERGREEN-PLAYER(F)) \geq fsat(F, all\ false)$$

We now prove that the loop invariant of AGGRESSIVE-EVERGREEN-PLAYER holds.

Proof. Observe that *oldratio* and *newratio* correspond to the satisfaction ratios of two consecutive formulae respectively, the latter being the *n-mapped* version of the former. We denote the former formula by F and the latter by F' . Therefore, we have $oldratio = fsat(F, EVERGREEN-PLAYER(F))$ and $newratio = fsat(F', EVERGREEN-PLAYER(F'))$.

By the property of the Evergreen player we just proved, we have

$$fsat(F', all\ false) \leq fsat(F', EVERGREEN-PLAYER(F')) = newratio$$

Also by the definition of *n-mapping*, we have

$$oldratio = fsat(F, EVERGREEN-PLAYER(F)) = fsat(F', all\ false)$$

Therefore, $oldratio \leq newratio$ holds throughout the loop.

6.3 *ELS*' Enforcement of Evergreen Laws

Interestingly, if we apply our local search algorithm to a given formula F and the *all false* assignment, $ELS(F, all\ false)$, it expands exactly to $AGGRESSIVE-EVERGREEN-PLAYER(F)$. Thus,

$$ELS(F, all\ false) = AGGRESSIVE-EVERGREEN-PLAYER(F)$$

This implies that *ELS* is a natural enforcer of the Evergreen laws. Therefore, for those MAX-CSP solvers that do not have the two properties that the Evergreen law requires, we can use *ELS* as their preprocessor in order to enforce the law.

7 Boosting MAX-CSP Solvers

We have implemented two preprocessors, one written in Scheme for boosting MAX-SAT solvers and the other written in Java for boosting MAX-CSP solvers.

We use the Scheme implementation as a preprocessor to boost the performance of an award-winning MAX-SAT solver, Toolbar[5], on MAX3SAT benchmarks from MAX-SAT Evaluation 2007. There are eight formulae in this benchmark. We allow Toolbar 20 minutes to run each of them. We compare the running time of Toolbar before and after the preprocessing on the four formulae upon which Toolbar finishes. Also, we compare the satisfaction ratio that Toolbar achieves before and after the preprocessing on the other four formulae upon which Toolbar times out. The experimental results in figure 2 illustrates the boosting effect of our preprocessor.

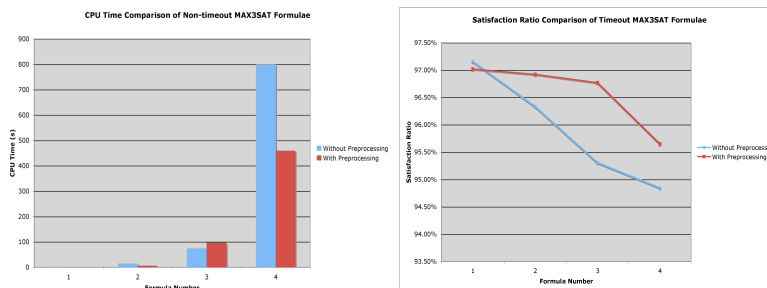


Fig. 2. Boosting Effect on MAX3SAT Formulae

We use the Java implementation as a preprocessor to boost the performance of another award-winning solver, Yices[6], on a formula containing 2000 variables and 8400 constraints (submitted by Oliver Kullman) from the SAT competition 2005. The hope of this preprocessing experiment is that the fast solver will notice that the assignment *all false* is pretty good and it will try to improve on it which should lead to good results

faster. Such a preprocessing experiment is a cheap way of blending the fast solvers with mean polynomials without having to modify the solver. The experimental result is very encouraging:
Yices without preprocessing:
running time = 888.048s, satisfaction ratio = 0.947143
Yices with preprocessing:
running time = 0.0342615s, satisfaction ratio = 1

8 Related and Future Work

Our Boolean MAX-CSP is a special case of Weighted Constraint Satisfaction Problem (WCSP). WCSP is an optimization version of the CSP framework in which constraints are extended by associating costs to tuples. Solving a WCSP formula consists of finding a complete assignment of minimal cost. Our Boolean MAX-CSP is a special case of WCSP because our domain is only Boolean and because the tuples can have only two costs: a negative cost when the relation is satisfied and a positive cost when the relation is unsatisfied. Several kinds of algorithms have been proposed to solve WCSP: Bucket Elimination [7] and several Branch and Bound algorithms (see [8] for an enumeration). None of those WCSP algorithms is using polynomials as abstract representations of WCSP problems. Although we present our results for Boolean MAX-CSP, the techniques also generalize to MAX-CSP, see section 8 of [1].

[9] discusses local search algorithms both for SAT and MAX-SAT and shows that local search outperforms complete algorithms on certain formulae. While traditional local search algorithms have a very simple neighborhood relation, our neighborhood relation is more complex but also efficiently computable. In addition, while traditional local search algorithms look for a largest increase or decrease within the neighborhood, we only find one point in the neighborhood. It should be noted, however, that we can also generate a large number of different assignments in the neighborhood. A random permutation of the variables in the formula is likely to lead to a different assignment when *mENF* or *ELS* is applied to the formula. Our basic neighborhood relation *EN* can be used in different ways to create local search algorithms, as suggested in [9] (e.g. a random walk strategy).

Preprocessing for SAT solvers is currently an active topic of research, e.g., [10]. Stochastic local search solvers may also benefit from a preprocessing phase borrowed from systematic SAT solving [11]. The kind of preprocessing we propose is novel, very different from resolution-based techniques.

Our motivation for working on MAX-CSP is that we think that it has important applications in biology and drug discovery. A recent paper from SRI confirms this conjecture [12] where MAX-SAT is used to analyze biological pathways.

9 Conclusions

Boolean MAX-CSP has numerous practical applications. For example, it serves as a flexible target language for many NP-hard optimization prob-

lems. Finding optimal strategies for the players of the Evergreen game reveals useful local search algorithms for boosting MAX-CSP solvers. We believe that the the Evergreen laws Loss-Free and Maximal are beneficial additions to the bag of tricks used in powerful Boolean MAX-CSP and WCSP solvers.

Acknowledgments: We would like to thank Ravi Sundaram for helping with the bound for the randomized algorithm, Bryan Chadwick for helping with implementing the preprocessor and Leonardo de Moura for his feedback on our preprocessor. We would also like to thank Novartis Institutes for Biomedical Research, Inc. for supporting this work. Karl Lieberherr spent his 2006 sabbatical at Novartis. Christine Hang is supported by a Novartis fellowship.

References

1. Lieberherr, K.J.: Algorithmic extremal problems in combinatorial optimization. *Journal of Algorithms* **3**(3) (1982) 225–244
2. Abdelmegeed, A., Hang, C.D., Rinehart, D., Lieberherr, K.J.: The Evergreen Game: The Promise of Polynomials to Boost Boolean MAX-CSP Solvers. Technical Report NU-CCIS-07-03, Northeastern University (2007) <http://www.ccs.neu.edu/research/demeter/biblio/evergreen.html>.
3. Borchert, B., Ranjan, D., Stephan, F.: On the computational complexity of some classical equivalence relations on boolean functions. *Theory of Computing Systems* **31** (1998) 679–693 <http://math.uni-heidelberg.de/logic/berichte.html>, Report 18.
4. Williamson, D.P.: Lecture notes on approximation algorithms. Technical Report RC 21409, IBM Research (1999)
5. Toolbar: . (<http://mulcyber.toulouse.inra.fr/projects/toolbar/>)
6. Yices: . (<http://yices.csl.sri.com>)
7. Dechter, R.: Bucket elimination: a unifying framework for processing hard and soft constraints. *ACM Comput. Surv.* **28**(4es) (1996) 61
8. Ansótegui, C., Bonet, M.L., Levy, J., Manyà, F.: The Logic Behind Weighted CSP. In Veloso, M.M., ed.: *IJCAI*. (2007) 32–37
9. Selman, B., Kautz, H.A., Cohen, B.: Local search strategies for satisfiability testing. In Trick, M., Johnson, D.S., eds.: *Proceedings of the Second DIMACS Challenge on Cliques, Coloring, and Satisfiability*, Providence RI (1993)
10. Anbulagan, Slaney, J.: Multiple Preprocessing for Systematic SAT Solvers. In: *IWIL-6*, as part of *LPAR-2006*, Phnom Penh, Cambodia (2006)
11. Anbulagan, Duc Nghia Pham, J.S., Sattar, A.: Boosting sls performance by incorporating resolution-based preprocessor. In: *Third International Workshop on Local Search Techniques in Constraint Satisfaction*, Springer Verlag (*LNCS* 244) (2006)
12. A. Tiwari and C. Talcott and M. Knapp and P. Lincoln and K. Laderoute: Analyzing Pathways using SAT-based Approaches. In: *Proc. 2nd Intl. Conf. on Algebraic Biology, AB 2007*. *LNCS*, Springer (2007)